Laser Wireframe Visualization

Kevin Chang

ECE 395 – ADVANCED DIGITAL PROJECTS LABORATORY
SPRING 2007

TA: Dan Barron

May 11, 2007

Abstract

The Laser Wireframe Visualization is an embedded system using an STR711 ARM microcontroller, designed to display a simple interactive wireframe model of a 3D object on a connected laser projector. It reads a description of the object from a file on an MMC card; processes the data to generate a perspective wireframe rendering, with a specified rotation applied to the object; and outputs the coordinates of a path tracing out the image, in the form of two analog voltages, to the laser projector.

Contents

1 Introduction **2**
1.1 Background and Motivation 2
1.2 Objectives 2
1.3 The STR711 ARM Microcontroller 3

2 Hardware **5**
2.1 Overview 5
2.2 STR-H711 Header Board 6
2.3 MMC Socket 6
2.4 DAC0808 Digital-to-Analog Converters and TL082 Dual Op-Amp 8
2.5 Other Components 8

3 Software **10**
3.1 Loading 3D Data from the MMC Card 10
3.2 Representing 3D Data and Rendering Projections 10
3.3 Source Files 12

4 Results and Future Work **14**

A Appendix: The Anglia IDEaliST IDE **16**
A.1 Setup and Creating a Project 16
A.2 Compiling, Downloading, and Debugging 17

1 Introduction

1.1 Background and Motivation

Constructing a homebuilt laser projector capable of displaying precise graphics is an impressive undertaking. A complex closed-loop system is required to make a laser draw anything more exact than rounded geometrical patterns, and only a couple records of successes can be found on the Web, where there is a small online community of enthusiasts dedicated to this hobby. One individual, known as ChaN, could probably be considered the pioneer of the subject; his page `http://elm-chan.org/works/vlp/report_e.html` provides the plans to build a projector system along with comprehensive background information. Another enthusiast named Norm is continually working to duplicate and improve on ChaN's design. He maintains an active discussion group for fellow laserists on his site, located at `http://24.202.226.162:81/LaserShow.htm`.

At the beginning of the Fall 2006 semester when I first read about these efforts, I also learned that the MacWarriors SIG of ACM@UIUC was working on building a projector for their LaserLine project (`http://www.acm.uiuc.edu/projects/LaserLine`). The goal of their project was to develop an application for Mac OS X to design laser shows and drive a laser projector, which they planned to build to ChaN's specifications. Knowing that there was a group nearby who I could work with was a major factor in my decision to pursue a related project.

1.2 Objectives

All of the laser controllers I had seen focused on displaying static two-dimensional line drawings. I decided that a controller with the ability to render a three-dimensional object as a wireframe would provide a more dynamic and interactive experience if the display were user-controlled. Alternatively, an expanded system could allow laser shows to be designed with much less work, as a designer could simply position and move 3D meshes instead of redrawing each frame separately. Being enrolled in CS 418 (Computer Graphics) in Fall 2006 contributed somewhat to my interest in 3D graphics.

I also planned the entire system to be self-contained as much as possible, meaning that it would have to be run by a microcontroller instead of via computer hookup. This would allow me to combine the controller and projector into one unit, making it more compact and portable, in addition to providing an opportunity for me to experience the development of an embedded system.

Finally, I wanted to ensure that the 3D object data could be easily loaded and swapped. This would require supporting some form of portable media, such as a memory card.

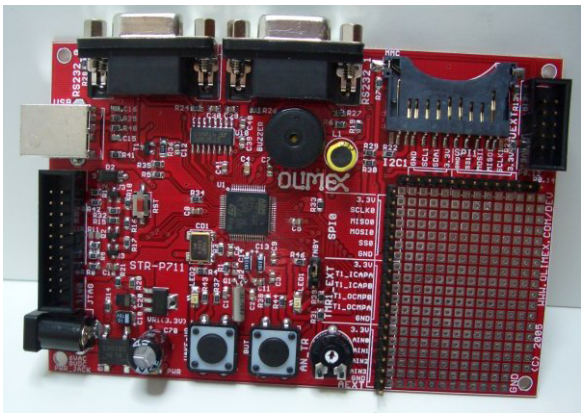

Figure 1: The STR-P711 development board (source: sparkfun.com)

1.3 The STR711 ARM Microcontroller

An Olimex STR-P711 development board featuring an STMicroelectronics STR711FR2 microcontroller, shown in Figure 1, was available in the ADSL lab. I concluded that the STR711 was the best platform for this project after exploring different options.

Two concerns were important when it came to the choice for the microcontroller: speed and available RAM. Storing and processing the data of a 3D object requires significant memory; the PIC16F87x series of microcontrollers used by several other ADSL projects provides a maximum of 368 bytes of data memory, which is not enough for this purpose (although it would be possible to connect an external RAM to a PIC). The STR711FR2 contains 64 kB of RAM, along with 256 kB of Flash memory for program code.

I also thought that the STR711 would be able to offer a performance advantage over a PIC. The STR711 is based on the 32-bit ARM architecture, as opposed to the 8- and 16-bit PICs. It can also execute up to 45 MIPS (million instructions per second) vs. a PIC16F87x's 5 MIPS, although the difference of architectures makes this a crude comparison. Regardless, it was my hope that the STR711 would better be able to handle the floating-point intensive calculations of a 3D transform, even though neither it nor the PICs has any dedicated floating point hardware.

The downside to the STR711 was that it had not been utilized by anyone in an ADSL project before; furthermore, there seemed to be very little information online detailing its use in hobbyist applications. Because of this, I had to learn over the course of this project the nuances of the STR711 chip's peripherals and the software library needed to control them. It was also necessary to find a toolchain capable of compiling, linking, and programming the chip. The majority of the tools available for ARM development are expensive commercial IDEs, and while there is a GNU ARM toolchain, I had trouble setting it up

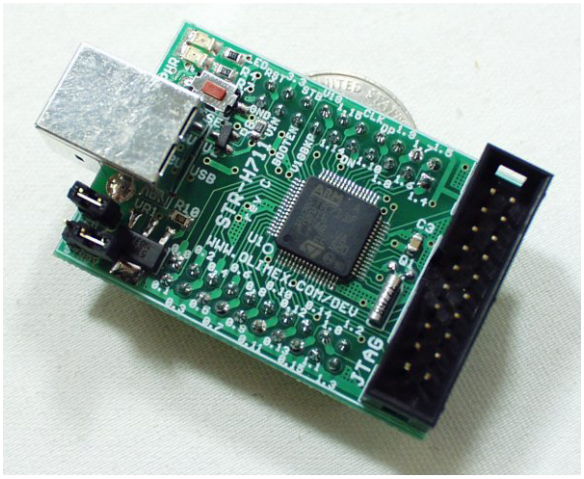

Figure 2: The STR-H711 header board (source: sparkfun.com)

with the proper linker scripts, etc. Fortunately, I was eventually able to find a free IDE (for Windows), Anglia IDEaliST, built on top of the GNU toolchain and geared specifically towards ST microcontroller development. The steps involved in setting it up are detailed in the Appendix.

Although the STR-P711 development board worked well for learning about and experimenting with the STR711, it contained many unnecessary external peripherals and its I/O pins were difficult to access. I therefore purchased an Olimex STR-H711 header board, as seen in Figure 2 and in Subsection 2.2, for the Laser Wireframe Visualization.

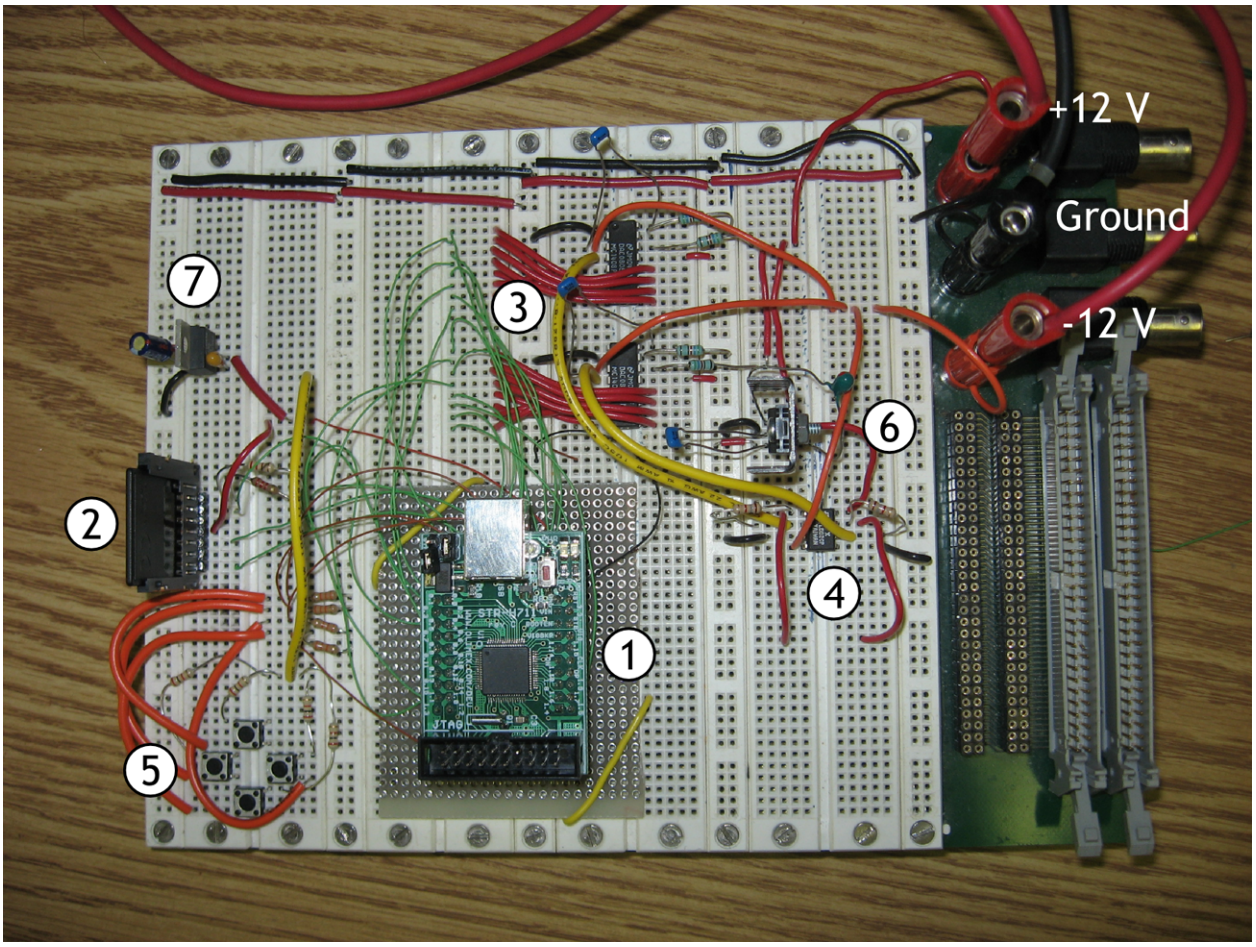

Figure 3: The Laser Wireframe Visualization system on a breadboard

2 Hardware

2.1 Overview

Figure 3 shows the Laser Wireframe Visualization system as implemented on a breadboard. The numbered components are:

1. STR-H711 header board
2. MMC socket and card
3. DAC0808 digital-to-analog converters
4. TL082 dual op-amp and analog outputs

5. Rotation buttons
6. 7805 12 to 5 volt regulator
7. LT1085 5 to 3.3 volt regulator

A schematic for the entire design can be seen in Figure 4.

2.2 STR-H711 Header Board

I chose to buy an Olimex STR-H711 header board (Figure 2) for the STR711FR2 from Spark Fun (`http://www.sparkfun.com/commerce/product_info.php?products_id=607`) because it allowed easier access to the STR711's I/O pins. This seemed like a better solution than a STR711FR2 chip alone because the header board was more breadboard-friendly and included support hardware such as a crystal and a JTAG programming header.

The header board is mounted on a pair of double row pin headers on a perf board (since inserting the headers directly in the breadboard would short adjacent pins). The I/O connections can be seen in the schematic (Figure 4). Several details of these connections are worth noting.

- The MMC card socket is connected to GPIO (General Purpose I/O) pins 0.4 through 0.7 because these are the pins for the STR711's second Buffered Serial Peripheral Interface (BSPI1). BSPI0 could also be used on P0.0-0.3, but I chose to duplicate the development board's setup, where the MMC socket was connected to BSPI1.
- GPIO pin 0.15 can only be used for input, as confirmed by page 24 of the STR71x datasheet. This detail caused me much trouble when I had P0.15 assigned as one of the DAC outputs until I became aware of the caveat.

The LED on the header board can be controlled by P1.8 if the corresponding jumper is set. It is toggled at several different points by my software to indicate status.

Schematics and details about the extension port assignments for the STR-H711 are available on Olimex's site at `http://www.olimex.com/dev/str-h711.html`. Information is also available for the STR-P711 development board at `http://www.olimex.com/dev/str-p711.html`. The datasheet and reference manual for the STR711 microcontroller itself can be found on ST's site at `http://mcu.st.com/mcu/modules.php?name=mcu&file=devicedocs&DEV=STR711FR2`.

2.3 MMC Socket

I decided to use an SD or MMC card for storage of the 3D object data, since the development board included an SD/MMC socket. Secure Digital, or SD, cards are commonly used in devices like digital cameras; MMC is SD's perhaps

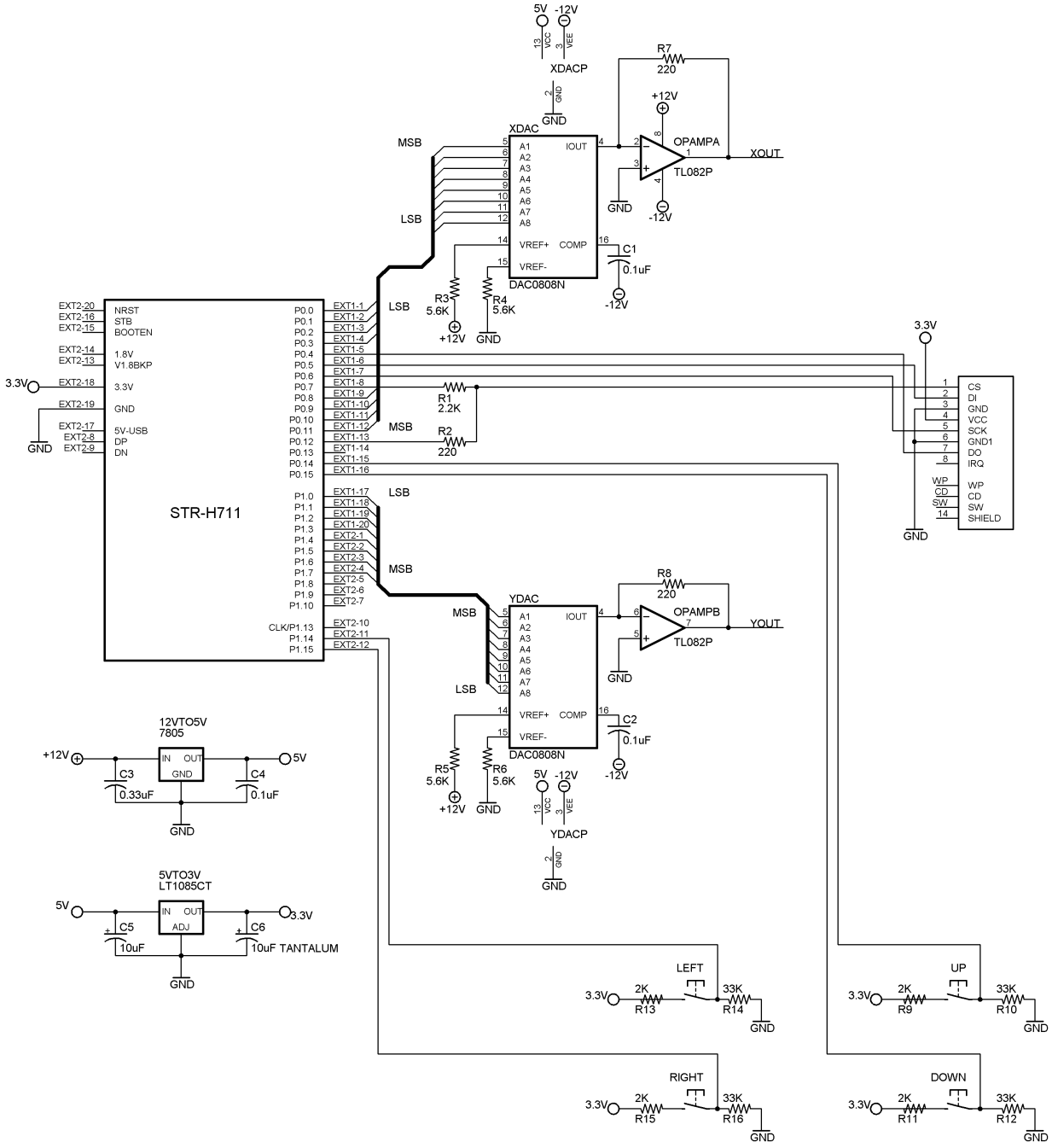

Figure 4: Schematic for the Laser Wireframe Visualization

less well-known predecessor, but it is compatible with SD for the most part. The first suitable card I found to use for this project was an MMC card, so the system has been designed with MMC in mind (see the Software section for more details).

The MMC socket was also purchased from Spark Fun (`http://www.sparkfun.com/commerce/product_info.php?products_id=136`). As mentioned before, it is connected to the BSPI1 interface of the STR711, where I duplicated the connections on the development board to ensure they were correct. However, the CS (chip select) signal on the MMC card is connected in a confusing way. Although the STR-P711 schematics on Olimex's site at `http://www.olimex.com/dev/str-p711.html` suggest that the SS1 signal from P0.7 on the STR711 is connected to CS, there is actually a 2.2 kΩ resistor between P0.7 and CS—and a 220Ω resistor between P0.12 and CS. In fact, the sample code provided by Olimex uses P0.12 to assert CS; while it may be possible to connect SS1 to CS directly, I copied the developtment board's arrangement to be on the safe side. Further details about the related software are covered in the Software section, in Subsection 3.1. (Thanks to DaveF on the Spark Fun forum for posting about this discrepancy; I added my findings to his topic at `http://forum.sparkfun.com/viewtopic.php?t=4992`.)

2.4 DAC0808 Digital-to-Analog Converters and TL082 Dual Op-Amp

A pair of DAC0808 digital-to-analog converters and a TL082 dual op-amp are responsible for converting 8-bit digital outputs from the STR711 into analog voltages for the laser projector, representing the displacements along the X and Y axes. In addition to the application notes in the DAC0808 datasheet, a lab handout (in English) from National Taiwan University at `http://www.bime.ntu.edu.tw/~ttlin/Course15/lecture_notes/C15_LAB_13.pdf`was very useful in helping me to determine the correct setup for the DACs and op-amp, especially the schematic on page 3.

I had no particular reason for choosing the TL082; many similar op-amps could probably be substituted. The resistors were selected to produce an output range of about 4 V, matching the range of ChaN's controller design, although there is not yet a provision for a +2 V reference voltage to shift the output to the -2 to +2 V range expected by the laser projector.

2.5 Other Components

The buttons to control rotation of the displayed 3D model are wired identically to the buttons on the development board, using a simple pull-down resistor design.

The 7805 voltage regulator converts the 12 V input from the breadboard to TTL 5 V for the DACs and op-amps. The LT1085 converts 5 V further down to CMOS 3.3 V for the STR711, MMC, and buttons. Details for implementing

both of these components can be found in their respective datasheets. I found that a small heat sink on the 7805 helped to keep it cooler; the LT1085 did not require one.

3 Software

3.1 Loading 3D Data from the MMC Card

I decided to read 3D object data from the MMC card in OBJ (sometimes referred to as Wavefront OBJ) format, a simple ASCII (text) format in widespread use among 3D applications. Anim8or (`http://www.anim8or.com`) is a free 3D modeling program, which can be used to edit and export OBJ files. The `load_mesh` function in `load.c` used to load OBJ files is adapted from CS 418 programming assignments.

In order to read from the MMC card, my system had to be able to deal with a FAT file system. Fortunately, not only is ChaN an authority on homebuilt laser projectors, he has also developed a module specifically for implementing FAT file systems on microcontrollers, called FatFs. I chose to use the lighter Tiny-FatFs version in this project since I would only be needing the most basic functions to open and read from a single file. Documentation for FatFs is available at `http://elm-chan.org/fsw/ff/00index_e.html`.

FatFs interfaces with functions for low-level disk access, which I had to provide in `mmc.c` and `diskio.c`. These functions were implemented by referring to ChaN's MMC/SD guide at `http://elm-chan.org/docs/mmc/mmc_e.html` and Olimex's sample code at `http://www.olimex.com/dev/str-p711.html`. Communication with the MMC card is done in SPI mode over the STR711's BSPI interface, as described earlier in the Hardware section (Subsection 2.3), instead of the faster native mode used by digital cameras and other commercial devices where read/write speed is important.

The procedures for working with SD and MMC cards in SPI mode are largely identical, according to ChaN's page; however, where the protocols differ, my implementation has been geared towards MMC. I have not tested the project with an SD card, although ensuring compatibility with both types of cards could probably be easily done.

3.2 Representing 3D Data and Rendering Projections

Modern computers almost invariably use APIs like OpenGL and Direct3D to render 3D graphics, usually with hardware acceleration. Because such an API would not be available to use on a microcontroller, I had to research graphics algorithms that worked directly with 3D source data to produce another set of data for display output. This meant looking for literature written on 3D graphics for computers one or two decades ago, before the advent of 3D graphics APIs. Two books, *Fast Algorithms for 3D-Graphics* by Georg Glaeser (New York: Springer-Verlag, 1994) and *3D Graphics in Pascal* by G. Bielig-Schulz and Ch. Schulz (Chichester: John Wiley & Sons, 1990), were invaluable references for developing my 3D processing procedures.

Glaeser's rendering approach involves storing a "pool" of the world coordinates of the 3D object's vertices, as read from the data file. Edge and face (polygon) pools store references to the vertices they contain in the world vertex

pool. To produce a perspective projection from a specified viewpoint, a matrix is constructed using viewpoint rotation parameters that transforms world coordinates to screen coordinates, and this matrix is applied to each of the world vertex coordinates to generate corresponding screen vertex coordinates. The edge and face pools can then refer to the screen vertex pool to specify how vertices on the screen are connected to form edges and faces. This is the approach I use in `draw.c` to apply a viewpoint transformation to the coordinates of a 3D object, except that I have eliminated the edge pool (combining it with the face pool) for better compatibility with the OBJ format's representation. Many of the supporting functions and macros in `vecmatop.h/.c` and `helpers.h` are also taken from Glaeser's book. For memory reasons, I limit the size of all pools to 255.

Because a wireframe rendering simply consists of lines connecting each of the vertices that form the edges of a face, drawing an image at this point would include lines on the back side of the object. There are various ways to deal with this problem, some of which only apply to raster displays like computer screens. (For example, polygons with the same color as the background may be drawn on top of the edges in the back of an object before the edges of the faces in front are highlighted.) For a vector output device like a computer plotter or a laser display, a hidden line removal algorithm must be used. Roberts's method as presented by Bielig-Schulz and Schulz is an algorithm suitable for this application, and I have ported their code from Pascal to C for my project, also in `draw.c`.

After the hidden lines have been removed, the remaining segments form an image that is ready to be output to the laser projector. To do this, I add the endpoints of each line passed by Roberts's algorithm to a list of points to draw, flipping the segment as necessary to share as many points as possible and removing duplicates of shared points. The resultant list is a path of points for the laser to trace, consisting of the segments to draw, where the laser should be on, as well as "jumps" connecting drawn paths, where the laser should be off. Another list keeps track of whether each segment is a visible drawn line or an invisible jump. There are actually two of each of these pools, so that the software can continue drawing an existing output path while a new one is being generated.

The actual output of the points on the path is directed by a timer interrupt, currently set to fire at about 16 kHz. Multiple stops are made along a visible segment as it is being drawn; to do this, the timer interrupt handler (`T3TIMI_IRQHandler`) in `71x_it.c` implements Bresenham's line algorithm, adapted from `http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html`. Each time the interrupt fires, a new point is calculated and output to the DACs.

I first developed and tested this rendering procedure in a Visual C++ project, which originated as an OpenGL application for CS 418. I gradually replaced OpenGL calls with my own methods until the entire algorithm (up to the generation of the output path) had been implemented, after which it was put in the microcontroller project as `draw.c`. I have presented only a basic overview of

the rendering process; refer to the code (which I tried to comment thoroughly) as well as the two books I used for better explanations.

3.3 Source Files

Here I will summarize the function of the source files included in the Anglia IDEaliST project.

`main.c`	Contains global definitions and coordinates program flow (initialization, data loading, and main program loop). The main loop polls the four rotation buttons, and upon detecting a press it redraws the image and swaps the working and output pools.
`defs.h`	Defines constants and data types.
`helpers.h`	Contains general macros and helper functions.
`globals.h`	Allows other source files to access global variables.
`init.h/.c`	Contains functions to initialize hardware clocks, the timer, the SPI interface, and the I/O ports.
`mmc.h/.c`	Contains functions to communicate with the MMC card over the SPI interface.
`diskio.c`	Wraps functions from `mmc.c` to implement low-level functions needed by ChaN's FatFs module (located in `fatfs/src`).
`tff.c`	The Tiny-FatFs version of ChaN's FatFs module (located in `fatfs/src`).
`load.h/.c`	Using Tiny-FatFs, looks for a file called `mesh.obj` on the MMC card and reads the data from it. I had to implement my own version of `fgets`, which I called `f_fgets`, since FatFs does not use standard streams. There is also a function `center_and_scale`, which centers the loaded object mesh around the origin and scales it to a standard size.
`draw.h/.c`	Contains the functions making up the processing and rendering procedure.
`vecmatop.h/.c`	Defines several macros and helper functions related to vector and matrix operations used by `draw.c`.
`startup.s`, `vector.s`	Assembly code provided by Anglia for startup and interrupt vector setup.

`rom71x.ld`	This file is not included as part of the IDEaliST project, but it resides in the project directory. It is the linker script provided by Anglia, which defines how the program and its data are laid out in the STR711's memory. I edited this file to increase the sizes of the stack and heap.

I tried to make sure my code was well-commented, so it should not be too difficult to interpret any of its meaning. The STR71x software library user manual available at `http://mcu.st.com/mcu/modules.php?name=mcu&file=devicedocs&DEV=STR711FR2` is a useful reference when dealing with the STR711's peripherals.

4 Results and Future Work

The Laser Wireframe Visualization is currently in a nearly finished state. While I have not tested it with a laser projector, its output can be demonstrated on an oscilloscope running in X-Y mode, as shown in Figure 5. However, many possible improvements could be made to this design:

- The output of the DACs is slightly glitchy; this can be seen in Figure 5 in the form of a series of dots to the left of the main image. I suspect this to be a timing issue with the DACs, which currently receive their inputs directly from the microcontroller. Latching the signals may solve the problem.
- There is occasionally a diagonal line crossing the entire image. This may be due to memory issues with the hidden line removal algorithm, which might be running out of stack space and throwing garbage values into the output pools. I believe the stack and heap sizes could be optimized further to deal with this problem.
- The addition of a +2 V output, possibly with a potentiometer, is probably necessary to interface with a laser projector built from ChaN's specifications.
- A blanking signal needs to be implemented in order to turn the laser off while it is jumping between visible segments.
- The output timing and spacing may need to be adjusted (this will depend on the properties of the connected projector). There may also be a better way to sort the segments in the output path.
- Connecting an external memory, if possible, would allow more complex objects to be loaded and drawn.
- A printed circuit board should be made for the finalized design for better durability and appearance.

Finally, the "other half" of this project could definitely be implemented—the construction of an actual laser projector to interface with the controller system. I have collaborated to this end with Joey Hagedorn, who was the main builder of the MacWarriors' laser projector; we had circuit boards fabricated and components purchased for the galvanometers (which position the laser beam) and their amplifiers, but time constraints forced me to focus on the development of my controller system, and I was unable to continue on my projector. Joey was able to go on and complete his projector, which MacWarriors showed with their LaserLine project at EOH 2007, though he had to substitute commercial amps when the homebuilt ones failed to work in time. I am interested in continuing work on the projectors with Joey in the future (probably Spring 2008, as I will be off campus during Fall 2007).

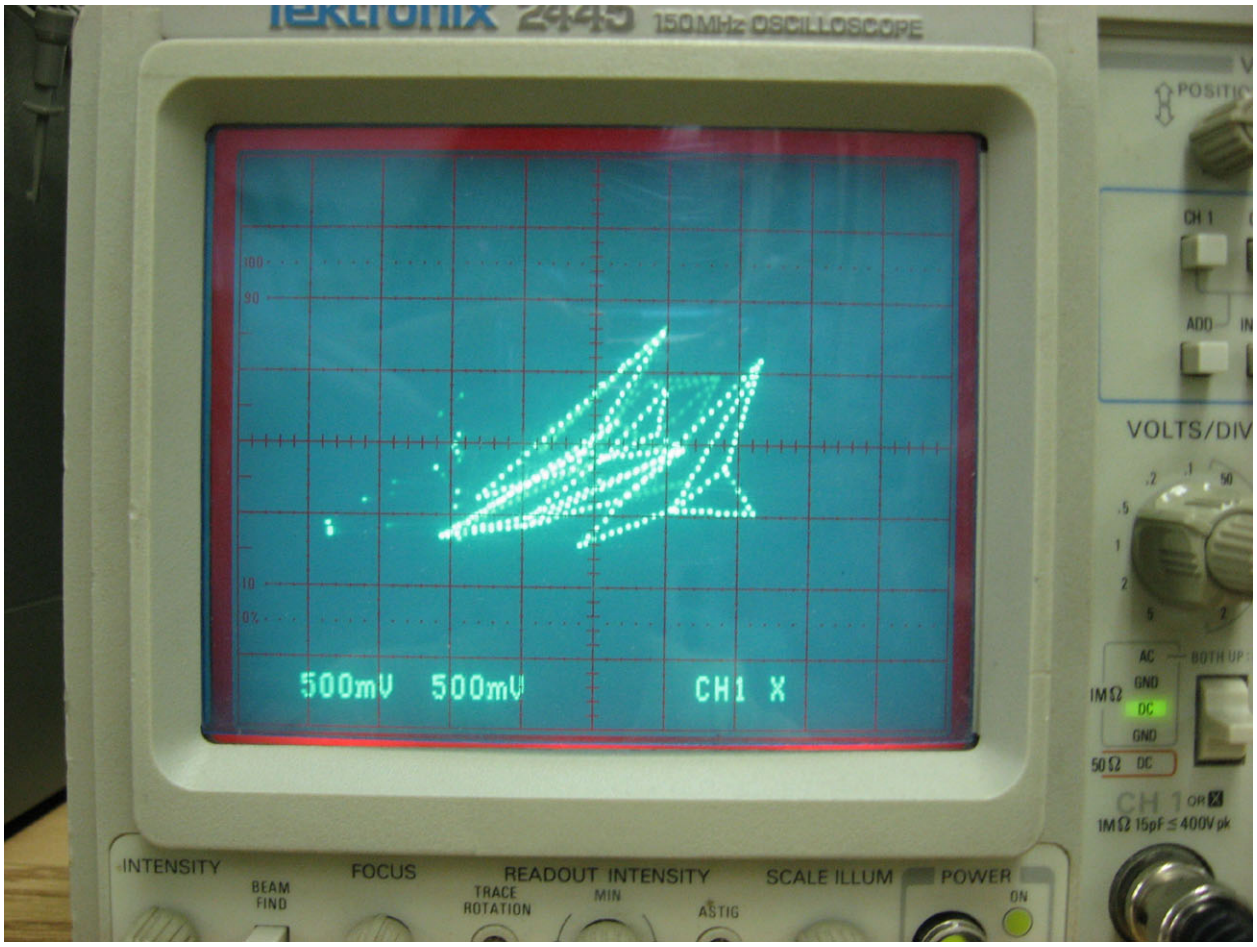

Figure 5: The Laser Wireframe Visualization system displaying a 3D model of an Arwing from Star Fox 64, sans wings (to reduce vertex/face count), on an oscilloscope.

A Appendix: The Anglia IDEaliST IDE

The Getting Started Guide available at `http://www.st-angliamicro.com/software.asp` provides some information about installing and using IDEaliST and the STR7 toolchain. Here I will provide a more comprehensive guide to setting up and working with the IDE, and some issues I encountered with it.

A.1 Setup and Creating a Project

1. Download the IDEaliST IDE and the STRx Toolchain from `http://www.st-angliamicro.com/software.asp`. I used the 4.1.1 toolchain.
2. Run both setup programs (the order doesn't matter). Make sure everything STR7-related is selected to be installed.
3. The first time IDEaliST is run, it will ask for a license, which you must request via email. In my experience, Anglia responds quickly (within a business day, but keep in mind they're UK-based).
4. Go to File ▷ New Project.... In the dialog, provide a name and location for your project. Select "STR7 Compiler (SARM GCC)" for the build tool and "STR711FR2T6" for the device. Leave the "Create Initial Project Files" box checked.
5. Copy `startup.s` and `vector.s` to your project directory from any of the example projects in `Anglia/IDEaliST/examples/str7`, and include them in the project by right-clicking one of the folders in the Workspace pane.
6. If you want to use interrupt handlers, you will have to include a file called `71x_it.c` in your project. This is most easily done by copying one provided by ST; unfortunately, it's not in any of the IDEaliST examples, but you can find one in the STR71x Standard Software Library download available at `http://mcu.st.com/mcu/modules.php?name=mcu&file=devicedocs&DEV=STR711FR2`.
7. Going to Options ▷ Project Settings... allows you to configure a couple things.
   - The Compiler Options tab includes settings like the optimization level and additional include folders.
   - The Linker Options tab is where you choose the linker script. This should be either "rom71x.ld" or "ram71x.ld", depending on whether you want to run the program from nonvolatile Flash ROM or RAM. (I didn't try running from RAM because of memory concerns, but it's supposedly better for debugging.) Assuming ROM is chosen, select "Pure Binary" from the "Create Other Format" dropdown as well, since that's the file that is written to Flash. Also check the "Use Math Libraries" box if you plan to use any math functions.

- The Debug Options tab can mostly be left alone. The programming cable available to me was a "Wiggler compatible" dongle (`http://www.olimex.com/dev/arm-jtag.html`), which is the default option.

8. You may need to edit the linker script, for example to change the stack and heap sizes like I did. To do this, copy the appropriate file (the one you selected in the Linker Options) from `Anglia/IDEaliST/examples/str7/scripts` to your project directory, and change it from there.

9. You can now begin adding source and header files and writing the code for your project.

One more thing: I can't stand "personalized menus" where infrequently-used menu items are hidden until you expand the menu. To turn this option off, go to View ▷ ToolBars ▷ Customize..., select the Options tab, and uncheck "Menus show recently used commands first."

A.2 Compiling, Downloading, and Debugging

The F7 key can be used to initiate a build. Upon success (when building for ROM), a `.bin` file will be generated. The program can be downloaded to the (powered and connected) STR711 by choosing Plugin ▷ Program Flash.... Be sure to select the correct file and reset the STR711 after programming (either manually on the board or with the button in the dialog) to start it running.

Alternatively, Plugin ▷ Run Insight Debugger... will automatically download the program and start the Insight Debugger. Details for using the debugger can be found in the Getting Started Guide. I didn't use the debugger much, since it didn't seem to be very stable and I couldn't set software breakpoints (this is a limitation of debugging from ROM, and I didn't learn how to use hardware breakpoints). However, it was useful in diagnosing a few software problems. One strange problem I ran into was that I couldn't view the contents of a variable after pausing execution unless it was a global variable defined in `main.c` (Insight would crash otherwise), so I debugged a few functions by making their variables global and accessing them with `extern`.